



Tina Linux 功耗管理 开发指南

版本号: 1.5.0
发布日期: 2024.10.23

版本历史

版本号	日期	制/修订人	内容描述
1.1.0	2021.01.03	AWA1610	适配新的文档模版
1.2.0	2021.07.11	AWA1610	1、对部分内容删减修正，并修改内容排布结构，按相似平台分类说明。 2、增加 R528，D1-H 平台说明。
1.2.1	2022.02.11	AWA1610	增加 R818B/MR813B 平台说明。
1.3.0	2022.03.11	AWA1610	1、增加 V853 平台说明。 2、删除部分冗余描述。 3、增加差异化说明，FAQ 常用问题章节。
1.3.1	2022.11.09	AWA1812	第一章 前言 1.3 小节增加 R853 平台说明
1.3.2	2023.04.25	AWA1812	第一章 前言 1.3 小节适用范围增加 MR527。 第五章 FAQ 5.4 小节增加”特定唤醒源无法唤醒系统“FAQ。
1.4.0	2023.11.03	AWA1812	第一章 前言 1.3 小节适用范围增加 AI985。 第一章 前言 1.4 小节增加文档约定。
1.4.1	2024.04.03	AWA1812	第一章 前言 1.3 小节适用范围增加 V851S3M100。
1.4.2	2024.08.21	AWA1812	第一章 前言 1.3 小节适用范围增加 MR536。 第二章 模块配置 2.2 小节增加内核配置项截图。
1.5.0	2024.10.23	AWA1812	适配新的文档模板，提供新版文档内容

目 录

1	前言	1
1.1	文档简介	1
1.2	目标读者	1
1.3	适用范围	1
1.4	文档约定	2
1.4.1	标志说明	2
1.4.2	地址与数据描述方法约定	2
1.4.3	数值单位约定	2
1.5	相关术语介绍	3
2	模块介绍	4
2.1	功能介绍	4
2.2	模块配置	5
2.2.1	Device Tree 配置说明	5
2.2.2	内核 menuconfig 配置说明	5
2.3	源码结构	8
2.4	软件流程	8
2.5	wakeup count 模块	10
2.6	wakelock 模块	10
3	功能开发	11
3.1	设备休眠唤醒回调注册	11
3.2	唤醒源注册	12
3.2.1	唤醒源分类	12
3.2.2	唤醒源支持列表	13
3.2.2.1	常见唤醒源	13
3.2.2.2	其他唤醒源	13
3.2.3	唤醒源说明	14
3.2.4	唤醒源注册方式	16
3.2.4.1	唤醒源注册相关接口	16
3.2.4.2	方法 1 在 driver probe 期间注册唤醒源	17
3.2.4.3	方法 2 在设备的休眠唤醒回调中使能/失能中断唤醒功能	18
3.3	休眠参考示例	18
4	调试方法	20
4.1	调试节点	20
5	FAQ	23
5.1	系统无法进入休眠流程	23

5.2 系统休眠后直接重启或延时几秒后重启	23
5.3 休眠后系统无法唤醒	24
5.4 特定唤醒源无法唤醒系统	24
5.5 一号通 FAQ 列表	25



插 图

图 2-1	功耗管理系统分类	4
图 2-2	Linux4.9 休眠唤醒配置示例	6
图 2-3	Linux5.4 休眠唤醒配置示例	7
图 2-4	Linux5.15 休眠唤醒配置示例	7
图 2-5	休眠唤醒回调顺序	9
图 3-1	中断结构	12



1 前言

1.1 文档简介

简要介绍 Tina 平台功耗管理机制，为关注功耗的开发者，维护者和测试者提供使用和配置参考。

1.2 目标读者

Tina 平台下关注功耗管理的广大客户、功耗管理相关的开发、维护及测试相关人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	休眠类型	参与功耗管理的协处理器
R328	Linux-4.9	NormalStandby	无
R818	Linux-4.9	SuperStandby	CPUS
R818B	Linux-4.9	SuperStandby	CPUS
MR813	Linux-4.9	SuperStandby	CPUS
MR813B	Linux-4.9	SuperStandby	CPUS
R329	Linux-4.9	SuperStandby	DSP
R528	Linux-5.4	NormalStandby	无
T113	Linux-5.4	NormalStandby	无
D1-H	Linux-5.4	NormalStandby	无
V853	Linux-4.9	SuperStandby/NormalStandby	无
R853	Linux-4.9	SuperStandby/NormalStandby	无
MR527	Linux-5.15	SuperStandby	CPUS
AI985	Linux-5.15	SuperStandby	CPUS
V851S3M100	Linux-5.15	SuperStandby/NormalStandby	无
MR536	Linux-5.15	SuperStandby	CPUS
V821	Linux-5.4	SuperStandby	RISCV 处理器

注：若同时支持多种休眠类型，则系统最终进入的休眠状态，通常根据唤醒源的配置自动确定。一般来说（无特别说明），唤醒源仅包括 rtc, PMU(powerkey、usb 插拔等) 这些唤醒源时，则进入 super standby，若存在其他任意的唤醒源，则进入 normal standby。

1.4 文档约定

1.4.1 标志说明

⚠ 注意

- 提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。

📖 说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

💡 技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-2: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200	地址或数据以 16 进制表示。
0b	0b010	数据采用二进制表示 (寄存器描述除外)。
X	00X	数据描述中，X 代表 0 或 1，例如，00X 代表 000 或 001。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

表 1-3: 数值单位约定

类型	符号	对应数值
数据容量	1 K	1024
数据容量	1 M	1 048 576

类型	符号	对应数值
数据容量	1 G	1 073 741 824
频率，数据速率等	1 K	1000
频率，数据速率等	1 M	1 000 000
频率，数据速率等	1 G	1 000 000 000

1.5 相关术语介绍

表 1-4: 术语表

术语	解释
CPUS	全志平台上，用于低功耗管理的协处理器单元。
CPUX	主处理器单元，主要为客户应用提供算力的 ARM/RISC-V 核心。
WFI	ARM 体系中一种指令，可将 CPUX 置于低功耗状态，直到有中断发生而退出该状态。详细请参考 ARM 手册，例如《DDI0487A_d_armv8_arm.pdf》。
NormalStandby、SuperStandby	Allwinner 内部术语，系统进入一种低功耗状态，暂停运行，以获取更低的功耗表现，区别是 CPUX 是否掉电。前者 CPUX 不掉电，系统唤醒直接借助于 CPUX 的 WFI 指令完成。后者 CPUX 掉电，系统唤醒需借助其他硬件模块实现，如 CPUS。
Arm Trusted Firmware	ARMv8-A 安全世界软件的一种实现，包含标准接口：PSCI、TBRR、SMCCC 等。在本文中，将其软硬件实现，统称为 ATF。
OP-TEE	一种安全操作系统方案，具有单独的 SDK 环境，以二进制文件的形式集成在 Tina 中，在本文中，统称为 OP-TEE。
SCP、ARISC	即 CPUS 的 SDK 环境。最初 CPUS 固件以闭源方式集成在 Tina 环境中，文件名为 scp.bin，故称 SCP。现已在 Tina 中提供开源代码包，目录名为 arisc，故又称为 ARISC。
BMU	电池管理芯片，提供电池升压，充电管理等功能，同时可外接电源键，用于开机，休眠，唤醒等。
PMU	电源管理芯片，有多个可调的 DC-DC, LDO 通道，提供电源管理功能，同时可外接电源键，用于开机，休眠，唤醒等。
wakelock	休眠框架提供的一种用户空间阻止系统进入休眠的机制，存在 wakelock 且使能检测时，系统休眠会被阻止。

2 模块介绍

2.1 功能介绍

Tina 功耗管理系统主要由休眠唤醒（standby、autosleep、runtime pm），调频调压（cpufreq、devfreq、dvfs），开关核（cpu hotplug），cpuidle 等子系统组成。主要用于对系统功耗进行管理和控制，平衡设备功耗和性能。

一般我们可将其分为两类，即静态功耗管理和动态功耗管理。

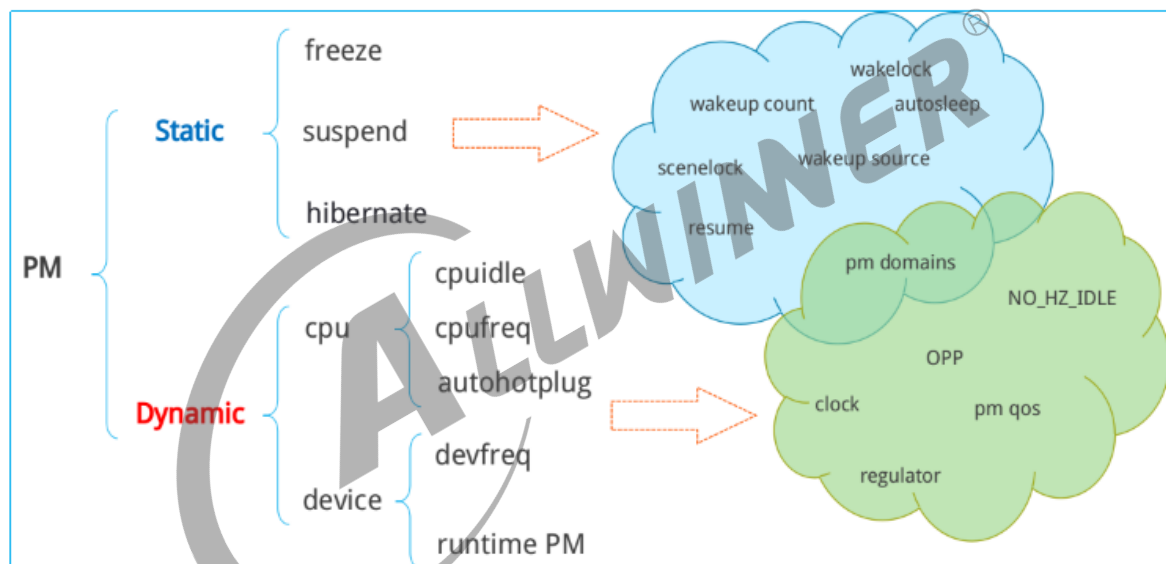


图 2-1: 功耗管理系统分类

一般地，可以动态调整或实时改变系统状态而达到节能目的的技术，被称为动态功耗管理，例如调频调压，idle, hotplug, runtime-pm 等。相对地，我们把单纯地将系统置为某一种状态，而不实时调整的低功耗技术，称为静态功耗管理，例如休眠唤醒相关技术等。

在 Tina 系统中，动态功耗管理已有默认配置，基本不需要修改，调频，温控等模块会在 Linux 模块开发指南目录下，由模块相关的文档说明。本文主要介绍静态功耗管理技术，即休眠唤醒框架。

2.2 模块配置

2.2.1 Device Tree 配置说明

standby_param 配置说明：

```
standby_param: standby_param {  
    vdd-cpu = <0x00000006>;  
    vdd-sys = <0x00000008>;  
    vcc-pll = <0x00000100>;  
    osc24m-on = <0x1>;  
};
```

“vdd-cpu = <0x00000006>”：各个 bit 代表 PMU 的各路供电，0x6 表示 vdd-cpu 使用 PMU 的第二、三路供电，休眠后第二、三路供电会被关闭；

“osc24m-on = <0x1>”：表示系统休眠后 osc24m 是否关闭，0x0 表示关闭，0x1 表示不关闭。

此处的配置值与系统休眠后的供电状态有关，影响系统的功耗与场景稳定性，一般不建议修改。

注：不使用PMU控制供电的平台没有此项配置

2.2.2 内核 menuconfig 配置说明

在 Tina 源码根目录，执行 make kernel_menuconfig，进入内核配置菜单。

如下图配置示例所示，进入 Power management options 配置项：

- Linux4.9:

```
.config - Linux/arm64 4.9.191 Kernel Configuration
> Power management options

Power management options
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded

[*] Suspend to RAM and standby
[ ] Skip kernel's sys_sync() on suspend to RAM/standby
[ ] Hibernation (aka 'suspend to disk')
[ ] Opportunistic sleep
[*] User space wakeup sources interface
(100) Maximum number of user space wakeup sources (0 = no limit)
[*] Garbage collector for user space wakeup sources
*- Device power management core functionality
[*] Power Management Debug Support
[*] Extra PM attributes in sysfs for low-level debugging/testi
[ ] Test suspend/resume and wakealarm during bootup
[ ] Enable workqueue power-efficient mode by default

<Select> < Exit > < Help > < Save > < Load >
```

图 2-2: Linux4.9 休眠唤醒配置示例

- Linux5.4:

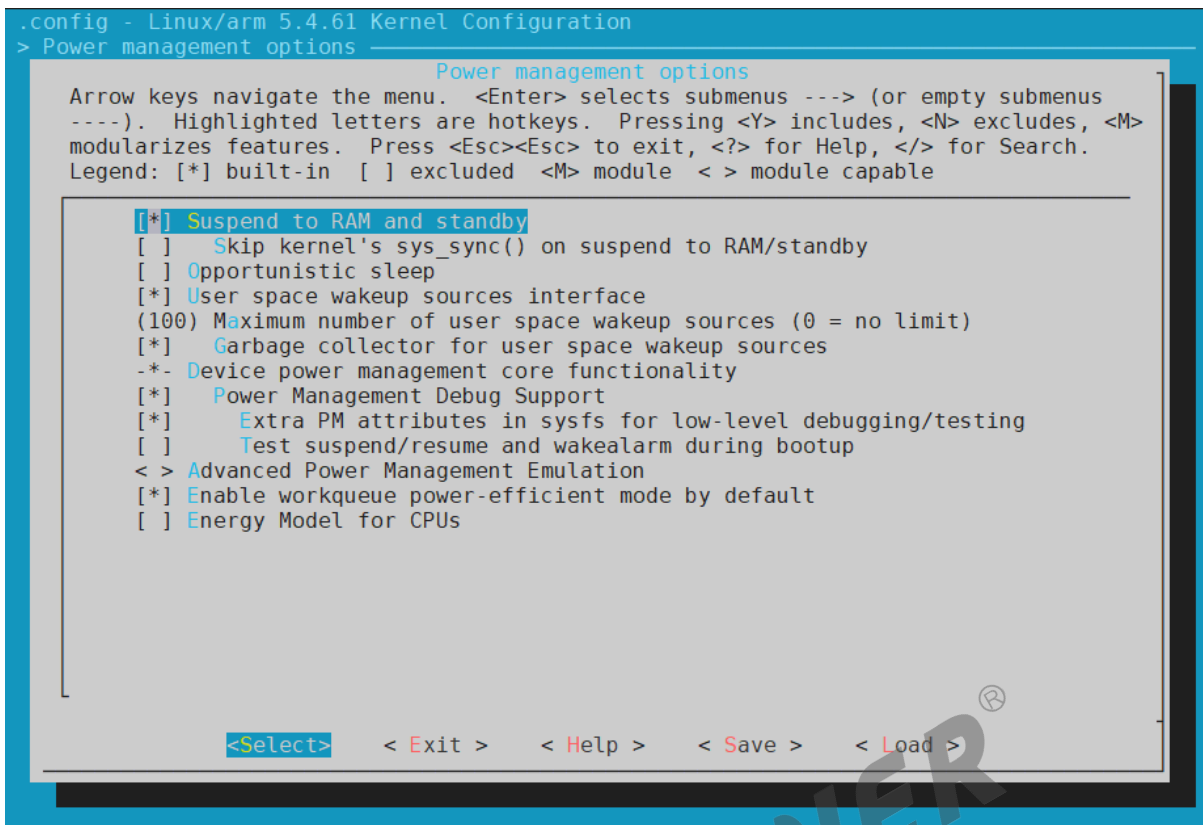


图 2-3: Linux5.4 休眠唤醒配置示例

- Linux5.15:

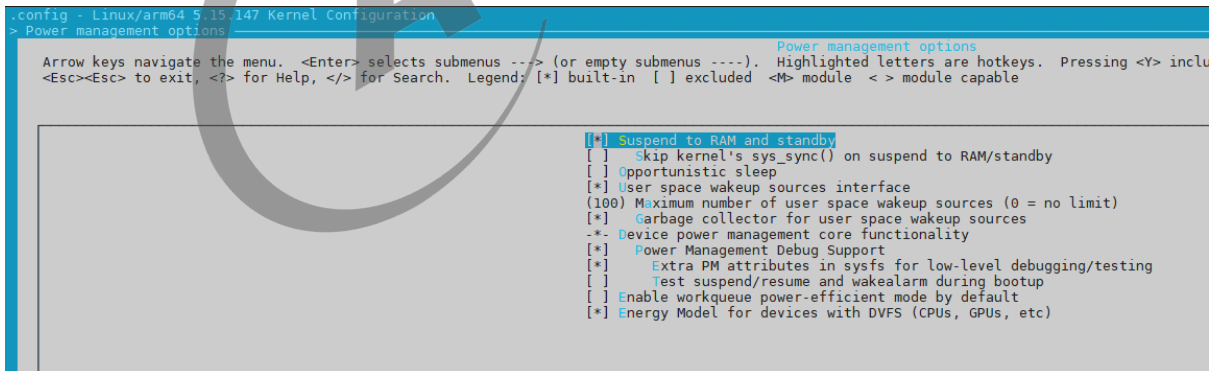


图 2-4: Linux5.15 休眠唤醒配置示例

使能以下关键宏:

```

SUSPEND=y # 使能suspend to ram休眠场景
PM=y # 使能PM框架
PM_WAKELOCKS=y # 可选, 使能wakelock机制
PM_DEBUG # 可选, 使能PM主要调节点
PM_ADVANCED_DEBUG # 可选, 使能更多调节点
    
```

2.3 源码结构

休眠框架核心代码位于内核根目录/kernel/power 中：

```
kernel/power/  
├── autosleep.c  
├── console.c  
├── energy_model.c  
├── hibernate.c  
├── Kconfig  
├── main.c # 文件节点  
├── Makefile  
├── power.h  
├── poweroff.c  
├── process.c  
├── qos.c  
├── snapshot.c  
├── suspend.c # 休眠主流程  
├── suspend_test.c  
├── swap.c  
├── user.c  
└── wakelock.c
```

2.4 软件流程

休眠唤醒流程基本上都是由内核框架完成，各家厂商差异不大。具体差异在于设备，系统，平台注册的回调函数，各厂商可通过修改这些回调，来适配各个平台，实现差异化。

内核主要休眠流程：

- 1、冻结用户进程和线程；
- 2、休眠控制台，同步文件系统；
- 3、休眠设备，调用设备休眠回调（prepare,suspend,suspend_late,suspend_noirq），内核根据唤醒源配置使能和关闭中断；
- 4、关闭非引导CPU，关闭全局中断；
- 5、调用syscore休眠回调，休眠系统服务，如kernel time等；
- 6、调用平台休眠回调（suspend_ops->enter），进入最终的休眠状态。在此阶段可关闭不必要的时钟，电源，并进入等待唤醒模式。Tina中，各平台最终休眠状态的差别在于此函数的实现。

内核主要唤醒流程：

- 1、检测到唤醒中断后开始平台唤醒，从平台休眠回调（suspend_ops->enter）中退出，并使能休眠时关闭的时钟，电源；
- 2、调用syscore唤醒回调，恢复系统服务；
- 3、使能全局中断，使能关闭的CPU；
- 4、恢复设备，调用设备唤醒回调（resume_noirq,resume_early,resume,complete），内核在此阶段还原中断配置；
- 5、恢复控制台；
- 6、恢复用户进程和线程，还原到休眠前的状态。

在整个休眠流程中，调用回调函数的顺序，如下图所示：

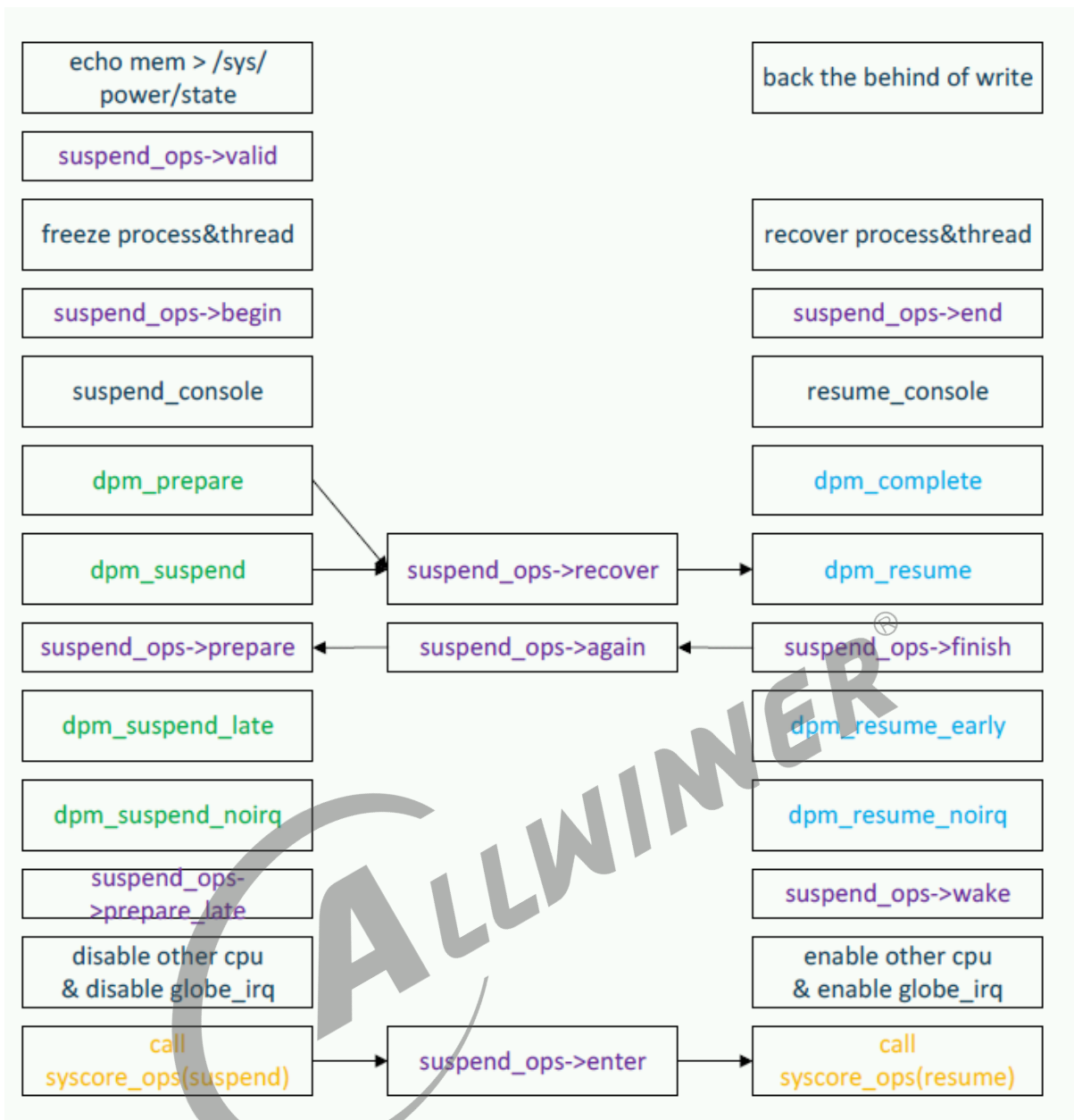


图 2-5: 休眠唤醒回调顺序

在本文中，无特殊说明，有如下约定：

绿色和蓝色方框部分：称为设备休眠唤醒回调，由设备驱动注册；每个驱动可注册一份或留空不注册，调用时，为每个设备都调用一次。

橙黄色方框部分：称为系统休眠唤醒回调，由内核模块注册，休眠系统服务，如内核时间服务等。

紫色方框部分：称为平台休眠唤醒回调，由平台厂商实现并注册，实现平台休眠逻辑，必须实现.valid 和.enter 函数，休眠的最终差异在于 enter 函数的实现不同。

2.5 wakeup count 模块

休眠唤醒是将系统从工作状态切换为非工作状态的一种技术，如果系统当前正在处理重要事件，而错误地切换到非工作状态，可能会造成使用体验不佳，甚至造成严重的系统问题。因此休眠唤醒系统需要保证系统在执行一些重要事件时，不能休眠。

因此，一个完整的休眠唤醒框架需要实现以下几点：

- (1) 当系统正在处理重要事件时，系统不可以进入休眠；
- (2) 系统休眠过程中，若发生了重要事件需要处理，休眠应立即终止；
- (3) 系统进入休眠状态后，若发生了重要事件需要处理，应当立即唤醒；

最终内核把上述的“重要事件”抽象为 wakeup event，为了解决上述问题，内核又实现了 wakeup count 模块。wakeup count 模块共维护两个计数，即系统当前正在处理的 wakeup event 个数 (inpr) 和系统已经处理完成的 wakeup event 总数 (cnt)。

休眠过程中，系统会比较 save_cnt (进入休眠时的 cnt 值) 和 cnt (当前系统的 cnt 值) 是否相同，并检测 inpr 是否为 0，若 cnt 发生变化或 inpr 不为 0，则内核会终止休眠。[®]

若要该模块生效，休眠前，发起休眠的程序需要对 PM 框架的 wakeup_count 节点写入当前的 cnt。

2.6 wakelock 模块

某些应用程序在执行特殊动作时可能需要阻止内核休眠，防止其他的应用程序或内核发起休眠导致设备或系统异常。

内核提供了 wake lock 模块，该模块通过 sysfs 文件系统向用户空间开放 wake_lock 和 wake_unlock 两个节点，应用程序可以通过这两个节点向内核请求一个 wakelock，此时内核会上报一个 wakeup event，修改 wakeup count 计数，阻止系统休眠。当应用程序处理完这一事件后，再通过 wake_unlock 节点释放对应的 wakelock，仅当系统中不存在任何一个 wakelock 时，系统才可以休眠。

若要该模块生效，休眠前，发起休眠的程序需要对 PM 框架的 wakeup_count 节点写入当前的 cnt。

3 功能开发

3.1 设备休眠唤醒回调注册

系统休眠唤醒会调用设备驱动自定义的休眠唤醒函数，将设备置入其自定义的低功耗状态，降低功耗以及避免设备在系统休眠状态下工作异常引发稳定性问题。

设备的休眠动作由驱动自定义，但从功耗与稳定性角度考虑，建议设备休眠时能够完成以下动作：

- (1) 停止设备驱动正在执行的动作：例如驱动有多个线程正在访问设备资源，应完成动作同步，停下动作或等待动作完成，并禁止下一轮动作发生；
- (2) 释放对时钟、电源的引用，失能中断；
- (3) 保存设备硬件状态：例如保存设备模块的寄存器，在唤醒后恢复，或在唤醒时重新初始化模块硬件；

休眠唤醒回调注册示例：

```
1. 第一步，定义dev_pm_ops结构体
#if IS_ENABLED(CONFIG_PM_SLEEP)
static struct dev_pm_ops case_pm_ops = {
    .suspend = case_suspend,
    .resume = case_resume,
};
#endif

2. 第二步，实现dev_pm_ops成员函数
#if IS_ENABLED(CONFIG_PM_SLEEP)
static int case_suspend(struct device *dev)
{
    ...
    return 0;
}

static int case_resume(struct device *dev)
{
    ...
    return 0;
}
#endif

static struct platform_driver case_driver = {
    .probe = case_probe,
    .remove = case_remove,
    ...
#if IS_ENABLED(CONFIG_PM_SLEEP)
```

```
.pm = &case_pm_ops,
#endif
...
};
```

3.2 唤醒源注册

3.2.1 唤醒源分类

唤醒源唤醒的本质是触发系统中断，因此在 Tina 平台上，我们可以按照中断不同将唤醒源分为两大类，

- 1、内部唤醒源，一般为 IC 内部外设，有自己独立的中断，如 RTC，UART，LRADC，USB 等。
- 2、外部唤醒源，这类设备都通过 GPIO 中断实现唤醒功能，占用一个对应的引脚，如 WiFi，BT，GPIOKEY 等。

如下图，粉色为 irq_chip 【GPIO 模块也看做是一个 irq_chip】，蓝色为内部唤醒源，紫色为外部唤醒源。

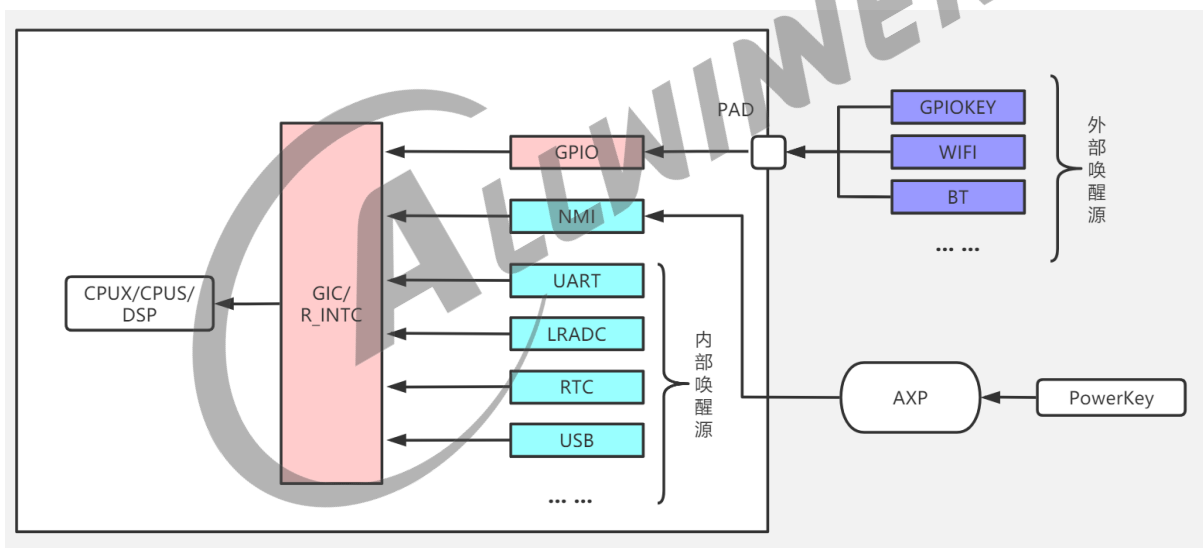


图 3-1: 中断结构

外部唤醒源不同于内部唤醒源，主要有以下不同：

- 1、外部唤醒源依赖于 GPIO 中断，而且 GPIO 中断通常是一个 GPIO Group 共用一个中断号，因此需要借助 irq_chip 框架进行虚拟中断映射。Tina 已经实现了映射，设备驱动使用 Linux 中断申请框架即可。
- 2、外部唤醒源使能唤醒功能时，还需设备驱动保证 GPIO 复用功能，时钟，电源，上下拉状态等正常。
- 3、GPIO 中断分为 CPUX 上的 GPIO 和 CPUS 上的 GPIO，以及 PMU 上的 GPIO，不同模块上的 GPIO 在实现上会有一些的差异，但 Tina 尽可能屏蔽了这些差异。

需要注意，不论哪种唤醒源，正常工作都有以下几个前提：

- 1、休眠后，发生预定事件后，设备可产生唤醒中断；由设备驱动在其 suspend/resume 函数中保证。
- 2、休眠后，该设备中断使能。设备驱动初始化或在 suspend/resume 函数中，向内核注册唤醒源，之后由休眠唤醒框架保证。

3.2.2 唤醒源支持列表

3.2.2.1 常见唤醒源

各平台上常见的唤醒源支持列表：

	PowerKey	LRADC	RTC	UART	USB 插拔	CPUX 域 GPIO	CPUS 域 GPIO
R328	N	Y	N	-	-	Y	N
R818	Y	N	Y	-	Y	N	Y
R818B	Y	N	Y	-	Y	N	Y
MR813	Y	N	Y	-	Y	N	Y
MR813B	Y	N	Y	-	Y	N	Y
R329	Y ¹	Y	Y	Y ²	-	N	Y
R528	N	Y	Y	-	-	Y	N
T113	N	Y	Y	-	-	Y	N
D1-H	N	Y	Y	-	-	Y	N
V853	Y	-	Y	-	Y	Y	N
R853	Y	-	Y	-	Y	Y	N
MR527	Y	-	Y	-	-	N	Y
AI985	Y	-	Y	-	-	N	Y
V851s3m100	-	-	Y	Y	-	Y	N
MR536	Y	-	Y	-	-	N	Y
V821	N	-	Y	N	N	N	N

注：Y: 支持；N: 不支持；-: 未明确

¹: 仅带 PMU 的方案支持。

²: 仅 suart 支持。

3.2.2.2 其他唤醒源

只在少数平台上支持的唤醒源列表：

	MAD	IC 内部 WiFi 模组	wakeup io	wakeup timer
R328	Y	N	N	N
R329	Y	N	N	N
V821	N	Y	Y	Y

注：Y: 支持；N: 不支持；-: 未明确

3.2.3 唤醒源说明

本节介绍 TinaLinux 各平台上主要使用到的唤醒源，简述其功能。由于各平台实现存在差异，对于以下唤醒源的支持可能不一致，具体请参考唤醒源支持列表。

- PowerKey

PowerKey（电源键）一般是连接在 PMU/BMU 上控制系统开机的按键，由 PMU/BMU 检测管理。当系统处于开机状态时，触发按键，则 PMU/BMU 会通过 NMI 中断上报按键事件。休眠框架，根据这个特性可支持其唤醒。

另外，PMU/BMU 也会通过 NMI 中断上报电池充电，电池过温等事件，由于这些事件都对应 NMI 中断，因此休眠框架无法区分，只能由 PMU/BMU 驱动控制使能。

一般地，在支持 PowerKey 的平台上，会默认使能此功能。

- LRADC 唤醒

利用 LRADC 按键模块，检测到按键后唤醒。

由于 LRADC 模块连接的多个按键对应一个 LRADC 中断，因此只能整体配置，无法单独禁用/启用某一个按键唤醒。

一般地，在 dts 中 keyboard 设备节点下，配置 “wakeup-source” 属性即可使能。

- RTC 唤醒

RTC 是日历时钟模块，其可以在关机，休眠等状态下正常走时，其支持设置一个未来时间点作为闹钟，当闹钟超时时，会产生 RTC 中断，触发系统唤醒。

下面提供一个配置 RTC 闹钟的方法，仅用于调试。量产产品中，应用程序应通过 /dev/rtc0 设备节点进行闹钟的配置，具体方法可参考 Linux 手册。

```
# 设置5秒后闹钟唤醒（注意定时时间从执行此条命令时开始计算）
echo +5 > /sys/class/rtc/rtc0/wakealarm
```

一般地，在 dts 中 rtc 设备节点下，配置 “wakeup-source” 属性即可使能。

- WiFi (GPIO) 唤醒

本质上是对应引脚的 GPIO 中断唤醒，IC 外部 WiFi 模组通过触发 GPIO 中断来唤醒系统。

依赖于 WiFi 模块本身对数据包的监听和管理，若模块或驱动无法支持，该功能亦无法使用，实际以模块自身配置为准。

一般地，模组驱动支持的情况下在 dts 的 WiFi 设备节点下，配置相应的 GPIO 引脚和 “wakeupsources” 属性即可使能唤醒功能，具体配置方法需查阅 TinaLinux WiFi 相关文档或与我司联系。

- BT (GPIO) 唤醒

与 BT 相同，本质上是对应引脚的 GPIO 中断唤醒。

依赖于 BT 模块本身对数据包的监听和管理，若模块或驱动无法支持，该功能亦无法使用，实际以模块自身配置为准。

一般地，默认未支持，具体配置方法，需查阅 TinaLinux BT 相关文档或与我司联系。

- UART 唤醒

通过 UART 接受到字符产生的中断，唤醒系统。

在 UART 唤醒功能中，有以下几点需要注意：

- 1、由于 UART 可能具有 FIFO，依赖于具体实现，可能不是每个字符都能产生中断并唤醒系统；
- 2、UART 一般需要至少 24MHz 以上的时钟频率，休眠需要保持时钟工作；
- 3、休眠唤醒系统只能识别到 UART 中断就立即唤醒，无法对数据包进行解析判断后唤醒；
- 4、有些平台，唤醒的动作由 CPUS/DSP 完成，因此存在 CPUS 与 CPUS/CPUS 分时复用 UART 设备的问题，导致数据丢失。

综上，我们不建议采用 UART 唤醒功能，如明确需要使用，可与我司联系，并评估上述问题风险。

一般地，默认未支持，具体配置方法，可与我司联系。

- USB 插拔唤醒

通过插拔 USB 时产生的中断唤醒系统。

这一般会依赖于 PMU 或 USB CC 器件支持，如明确需要使用，可与我司联系。

一般地，默认未支持，具体配置方法，需查阅 TinaLinux USB 相关文档或与我司联系。

- MAD 唤醒

休眠后依靠硬件检测语音信号能量，若超过预设的阈值，将产生 MAD 中断唤醒系统且同步录音。

一般地，默认未支持，具体配置方法，需查阅 TinaLinux 音频相关文档或与我司联系。

- GPIO 唤醒

通过触发中断复用功能的 GPIO 引脚中断来唤醒系统。需要驱动模块完成对 GPIO 的配置，并为使用的 IO 中断注册唤醒源。

一般地，GPIO 根据所在的电源域，分别 CPUX 域 GPIO 与 CPUS 域 GPIO，super standby 场景下仅支持 CPUS 域的 GPIO 唤醒，而 normal standby 场景下两个域的 GPIO 唤醒均支持。

- IC 内部 WiFi 模组

IC 内部的 WiFi 模组在休眠后保持工作，产生独立的中断来唤醒系统。

一般地，在支持内部 WiFi 模组的平台上默认使能。

- wakeup io

wakeup io 是一个独立的模块，有自己的独立中断，依赖指定 IO 配置为输入模式作信号检测，IO 电平保持高或低指定时间后触发模块中断。

wakeup io 通常位于 RTC 电源域，包含该模块的平台在各休眠模式下都能使用 wakeup io 唤醒系统，配置方法的具体配置方法可与我司联系。

- wakeup timer

位于 RTC 电源域的定时器，用于短时间定时。

一般地，在支持 wakeup timer 的平台上默认使能。

3.2.4 唤醒源注册方式

3.2.4.1 唤醒源注册相关接口

以下注释来自Linux-5.4内核，4.9、5.4、5.15内核的注册方式相同：

```
1.
头文件：include/linux/pm_wakeup.h
/**
 * device_init_wakeup - Device wakeup initialization.
 * @dev: Device to handle.
```

```

* @enable: Whether or not to enable @dev as a wakeup device.
*
* By default, most devices should leave wakeup disabled. The exceptions are
* devices that everyone expects to be wakeup sources: keyboards, power buttons,
* possibly network interfaces, etc. Also, devices that don't generate their
* own wakeup requests but merely forward requests from one bus to another
* (like PCI bridges) should have wakeup enabled by default.
*/
int device_init_wakeup(struct device *dev, bool enable)

2.
头文件: linux/pm_wakeirq.h
/**
* dev_pm_set_wake_irq - Attach device IO interrupt as wake IRQ
* @dev: Device entry
* @irq: Device IO interrupt
*
* Attach a device IO interrupt as a wake IRQ. The wake IRQ gets
* automatically configured for wake-up from suspend based
* on the device specific sysfs wakeup entry. Typically called
* during driver probe after calling device_init_wakeup().
*/
int dev_pm_set_wake_irq(struct device *dev, int irq)

3.
头文件: linux/interrupt.h
static inline int enable_irq_wake(unsigned int irq)
{
    return irq_set_irq_wake(irq, 1);
}

static inline int disable_irq_wake(unsigned int irq)
{
    return irq_set_irq_wake(irq, 0);
}
/**
* irq_set_irq_wake - control irq power management wakeup
* @irq: interrupt to control
* @on: enable/disable power management wakeup
*
* Enable/disable power management wakeup mode, which is
* disabled by default. Enables and disables must match,
* just as they match for non-wakeup mode support.
*
* Wakeup mode lets this IRQ wake the system from sleep
* states like "suspend to RAM".
*/
int irq_set_irq_wake(unsigned int irq, unsigned int on)

```

3.2.4.2 方法 1 在 driver probe 期间注册唤醒源

在 driver probe 时, 通过 device_init_wakeup() 接口获得唤醒源能力, 再使用 dev_pm_set_wake_irq() 接口将中断设置为唤醒中断, 操作示例:

```

if (of_property_read_bool(pdev->dev.of_node, "wakeup-source")) { //检查dts中该设备配置了wakeup-source
    device_init_wakeup(&pdev->dev, true); //获取唤醒能力
    dev_pm_set_wake_irq(&pdev->dev, irq_number); //注册唤醒中断
}

```

}

注：当设备作为唤醒源时，设备的休眠回调不应执行完全休眠该设备的动作，需要保留该设备唤醒相关的能力及依赖资源。

此后每次休眠，休眠框架会为设备使能、失能中断唤醒功能。

3.2.4.3 方法 2 在设备的休眠唤醒回调中使能/失能中断唤醒功能

操作示例：

1. 第一步，在 driver probe 时，通过 device_init_wakeup() 接口获得唤醒源能力，并在驱动内部设置标识表明自身被配置为唤醒源：

```
if (of_property_read_bool(pdev->dev.of_node, "wakeup-source")) { //检查dts中该设备配置了wakeup-source
    device_init_wakeup(&pdev->dev, true); //获取唤醒能力
    priv_data->wakeup = 1; //表明自身有被配置为唤醒源
}
```

2. 第二步，当需要使能唤醒功能时，在设备的休眠唤醒回调中，调用 enable_irq_wake()/disable_irq_wake() 接口使能/失能唤醒中断：

```
static int case_device_suspend(struct device *dev)
{
    if (priv_data->wakeup && (如有必要，判断设备需要用来唤醒)) {
        保留设备产生中断的功能及所需资源
        enable_irq_wake(irq_number);
    } else {
        休眠该设备
    }
}

static int case_device_resume(struct device *dev)
{
    if (priv_data->wakeup && (如有必要，判断设备需要用来唤醒)) {
        disable_irq_wake(irq_number);
    } else {
        唤醒该设备
    }
}
```

注：当设备作为唤醒源时，设备的休眠回调不应执行完全休眠该设备的动作，需要保留该设备唤醒相关的能力及依赖资源

3.3 休眠参考示例

- 1、首先读出当前系统的 wakeup count

若读取时阻塞，说明系统存在 wakeup event 正在处理，即 inpr 不为 0，此时不能休眠。

若读取成功，则说明 inpr 为 0，且读出的值即为系统当前的 cnt。

```
root@TinaLinux:/# cat /sys/power/wakeup_count
8
```

2、将读出的 cnt 写回 wakeup_count

若写入成功，说明 cnt 被内核保存为 save_cnt，之后系统可以休眠。

若写入失败，说明在本次读写 cnt 的过程中产生了 wakeup event，应该重复步骤 1~2，直到写入成功。

```
root@TinaLinux:/# echo 8 > /sys/power/wakeup_count
```

3、尝试休眠

若休眠过程中未产生 wakeup event，系统成功休眠。

若休眠过程中产生了 wakeup event，内核会检测到 inpr 不为 0，或当前 cnt 不等于 save_cnt，系统会终止休眠，回退到正常状态，应用程序可等待一段时间后，重复 1~3 步，再次尝试。

```
root@TinaLinux:/# echo mem > /sys/power/state
```

休眠脚本示例：

```
#!/bin/ash

function suspend()
{
    while true; do
        if [ -f '/sys/power/wakeup_count' ]; then
            cnt=$(cat /sys/power/wakeup_count)
            echo "Read wakeup_count: $cnt"
            echo $cnt > /sys/power/wakeup_count

            if [ $? -eq 0 ]; then
                echo mem > /sys/power/state
                break;
            else
                echo "Error: write wakeup_count($cnt)"
                sleep 1;
                continue;
            fi
        else
            echo "Error: File wakeup_count not exist"
            break;
        fi
    done
}

echo "try to mem..."
suspend
```

💡 技巧

休眠时不应连接 usb，在 usb 连接状态下，usb 驱动会上报 wake event 且不会释放，这会导致读取 wakeup_count 阻塞。若出现执行阻塞的情况，请尝试拔掉 USB。

4 调试方法

4.1 调试节点

💡 技巧

若需要使用调试节点进行 debug 或测试，请使能以下内核宏：PM_DEBUG PM_ADVANCED_DEBUG

state

路径：/sys/power/state

Linux 标准节点，系统休眠状态配置节点。通过写入不同的状态级别（freeze，standby，mem）可使系统进入到不同级别的休眠状态。

freeze 状态为 Linux 系统自身支持的一种休眠状态，与平台无耦合，不调用到平台回调接口，无底层总线，时钟，电源控制，但会在调用设备休眠回调后进入 cpuidle 状态。

standby，mem 状态在 Tina 中效果相同。

```
# 强制进入休眠，不会判断系统 inpr, cnt 状态
root@TinaLinux:/# echo mem > /sys/power/state
```

⚠ 注意

未通过 wakeup_count 节点判断系统当前状态是否可以休眠，而直接使用 echo mem > /sys/power/state 命令强制系统进入休眠会使休眠唤醒流程忽略对 inpr 和 cnt 变量检测，可能会导致一些同步问题。如休眠过程中，WiFi 唤醒中断不能导致休眠流程终止，而出现系统强制休眠，无法唤醒的异常。

wakeup_count

路径：/sys/power/wakeup_count

Linux 标准节点，将 wakeup count 模块维护的计数开放到用户空间，为应用程序提供一个判断系统是否可以休眠的接口。

具体使用参考上文 wakeup count 相关说明。

wake_[un]lock

路径：/sys/power/wake_lock、/sys/power/wake_unlock

Linux 标准节点，wake lock 模块开放到用户空间的接口。

应用程序可以通过 wake_lock 节点申请一个 lock，并通过 wake_unlock 节点释放对应的 lock，

任一应用程序持有 wakelock，系统都不休眠。

```
# 申请一个NativePower.Display.lock
root@TinaLinux:/# echo NativePower.Display.lock > /sys/power/wake_lock
# 可以查看有系统中存在哪些wakelock
root@TinaLinux:/# cat /sys/power/wake_lock
NativePower.Display.lock

# 释放 NativePower.Display.lock
root@TinaLinux:/# echo NativePower.Display.lock > /sys/power/wake_unlock
# 可以查看那些wakelock被释放
root@TinaLinux:/# cat /sys/power/wake_unlock
NativePower.Display.lock
```

💡 技巧

注意：强制休眠命令不会判断系统 inpr, cnt 状态，因此 wake_lock 机制无效。

pm_print_times

路径：/sys/power/pm_print_times

Linux 标准节点，该节点标志是否在休眠唤醒流程中，打印 device 休眠唤醒调用信息。

该节点默认值为 0，即不打印设备调用信息。

```
# 使能设备回调信息输出
root@TinaLinux:/# echo 1 > /sys/power/pm_print_times
```

pm_wakeup_irq

路径：/sys/power/pm_wakeup_irq

Linux 标准节点，只读。用于查看上一次唤醒系统的唤醒中断号。

📖 说明

在 Linux-4.9 中，该节点对于外部唤醒源的中断无法正常显示。

这是由于 pinctrl 驱动中，为 gpio 设置了 IRQF_NO_SUSPEND 标志导致，由于影响模块较多，暂不处理。

```
# 使能设备回调信息输出
root@TinaLinux:/# cat /sys/power/pm_wakeup_irq
```

pm_test

路径：/sys/power/pm_test

Linux 标准节点。由内核实现的一种休眠唤醒调试机制。

读该节点会打印其支持的调试点，如下：

```
# linux 默认支持的调试点
root@TinaLinux:/# cat /sys/power/pm_test
[none] core processors platform devices freezer
```

对该节点写入其支持的调试点，会在休眠过程中，执行到该调试点时，等待几秒后返回。

```
root@TinaLinux:/# echo core > /sys/power/pm_test
```

📖 说明

Freezer: 任务冻结后, 等待 5s, 即返回;
Devices: 执行设备回调 prepare, suspend 后, 等待 5s, 即返回;
Platform: 执行设备回调 suspend_late、suspend_noirq 后, 等待 5s, 即返回;
Processors: 关闭非引导 cpu 后, 等待 5s, 即返回;
Core: 冻结系统服务, 如内核时间服务后, 等待 5s, 即返回;
None: 整个休眠流程全部走完, 需触发唤醒源唤醒;

console_suspend

路径: /sys/module/printk/parameters/console_suspend

Linux 标准节点, 该节点标记在系统进入休眠时, 是否休眠控制台。

这个节点默认值为 Y, 即默认会休眠控制台。

将其设置为 N 后, 系统休眠时将不休眠控制台, 这样可以将休眠后期 (控制台休眠阶段后) 的日志实时打印到控制台, 便于调试。

```
# 禁用控制台休眠  
root@TinaLinux:/# echo N > /sys/module/printk/parameters/console_suspend
```

ignore_loglevel

路径: /sys/module/printk/parameters/ignore_loglevel

Linux 标准节点, 忽略打印级别控制。

这个节点默认值为 N, 即不忽略打印级别, 仅输出可打印级别的日志。可打印级别由 proc/sys/kernel/printk 点控制。

将其设置为 Y 后, 任何级别的系统日志都可以输出到控制台。这不仅仅在休眠唤醒过程中有效, 在系统正常工作时也有效。

```
# 忽略系统日志打印级别  
root@TinaLinux:/# echo Y > /sys/module/printk/parameters/ignore_loglevel
```

initcall_debug

路径: /sys/module/kernel/parameters/initcall_debug

Linux 标准节点, 该节点标记是否开启内核早期日志, 在内核启动早期先初始化控制台, 输出内核启动早期日志信息。在休眠唤醒流程中, 会影响到唤醒早期部分日志的打印。

该节点默认值由内核参数确定, 一般为 N, 即不使能早期打印。将其设置为 Y 后, 会多打印 syscore_ops 调用信息。

使能该节点后, 会休眠唤醒过程中打印各个设备休眠唤醒回调的调用顺序及返回值, 通过这些打印信息, 可以判断出是哪个设备休眠唤醒回调出了问题, 方便调试。

```
# 使能早期打印  
root@TinaLinux:/# echo Y > /sys/module/kernel/parameters/initcall_debug
```

5 FAQ

5.1 系统无法进入休眠流程

一般是由于使用了 wakelock 机制，在休眠前判断系统状态时，系统存在 wakelock，最终导致系统无法进入休眠流程。

处理：

- 通过 `cat /sys/power/wake_lock` 来确认是否存在 wakelock，若存在 wakelock，可以通过 `echo xxx > /sys/power/wake_unlock` 来取消该 wakelock；
- 如果连接了 usb，usb driver 可能会申请 wakelock 或上报唤醒事件，阻止休眠，wake_lock 节点无法读出来，可以尝试拔掉 usb。
- 可以直接执行 `echo mem > /sys/power/state` 在不释放 wakelock 的情况下，强制休眠，来验证是否被机制阻止休眠。建议此操作仅用于临时调试，因为该操作会导致 wakelock 没有效果。

5.2 系统休眠后直接重启或延时几秒后重启

这种问题一般是由于休眠过程中，某一驱动模块 oops 卡死，导致触发保护机制重启，或休眠后系统掉电异常，例如 rtc 的电也掉了导致。

对于前者，可以使能休眠唤醒日志，确认是哪一模块，然后找相关同事协助处理；临时测试，也可以先尝试去掉该驱动模块；

```
# 使能休眠唤醒所有的日志
echo 1 > /sys/power/pm_print_times;
echo N > /sys/module/printk/parameters/console_suspend;
echo Y > /sys/module/kernel/parameters/initcall_debug;
echo 8 > /proc/sys/kernel/printk;
```

对于后者，先排除前者问题后，可以用万用表或示波器抓取一些关键电源的休眠状态，如 vcc-rtc, vdd-cpu, vdd-sys, vcc-pll 等，然后与正常机器比较，或找相关硬件同事确认。

也有一些其他原因，如内存踩踏等，可导致此现象，这里不展开说明。

5.3 休眠后系统无法唤醒

这种问题是最常见的休眠唤醒问题，导致该现象的问题原因也比较多，包括但不限于，唤醒源配置有误，内核卡死但未触发重启，cpus/dsp/optee 等卡死，内存踩踏或使用超出范围内存，dram 不正常，系统硬件问题。一般建议通过以下流程逐步收集一些有用信息：

1. 使用 powerkey, rtc 等默认唤醒源唤醒，排除由于唤醒源配置导致的无法唤醒；
2. 使能日志，排除由于系统卡死，导致休眠没有完成而表现出无法唤醒的现象；
3. 与正常机器对比，回退部分提交，确认问题大致什么时间以及什么模块引入；
4. 通过 /sys/power/pm_test 节点，执行不同深度的休眠，确认问题点出现在休眠唤醒流程的哪个阶段；
5. 如果 echo core > /sys/power/pm_test 后仍不可以唤醒，说明问题大概率出现在 kernel 模块中，否则问题可能在 cpus/dsp/optee 等阶段；
6. 通过观察 cpus 串口日志，确认其是否存活。
7. 通过仪器测量各路电源状态，以及在休眠流程中对一些寄存器（时钟、电源、IO 状态）值进行确认，细化问题点；
8. 如果上述都不能找到有效点，可以联系全志处理，并尽可能提供相关信息。

5.4 特定唤醒源无法唤醒系统

发现系统可正常唤醒，但却无法使用特定的唤醒源来唤醒，例如新增了一个设备使用 GPIO 唤醒失败，可按如下思路排查原因：

首先确认唤醒源在休眠后能够工作，例如：测量是否保留供电、确认驱动未在休眠时关闭自身工作条件（如在休眠回调内执行了关时钟、关中断等操作）。

然后检查唤醒中断通路是否导通，以 gpio 中断为例：

1. 休眠前，检查中断是否正常产生、中断处理是否触发；
2. 休眠时，设备休眠后打印寄存器值，或系统完全休眠后使用参与功耗管理的协处理器查看寄存器值，检查：（1）IO 是否使能；（2）IO 中断是否使能；（3）休眠后是否能产生 pending；

若以上 3 个条件均满足，且系统未死机，较大可能是唤醒源的配置有误，需要进一步检查唤醒源的配置方式。

5.5 一号通 FAQ 列表

更多 FAQ 请登录 allwinner 一号通平台，在常见问题中搜索“休眠唤醒”关键词获取。

典型 FAQ：

- (1) 【FAQ3088】Linux 休眠唤醒数据被修改时定位执行修改操作的设备
- (2) 【FAQ2801】Linux 设有 IRQF_NO_SUSPEND 的中断唤醒未被记录
- (3) 【FAQ2080】Tina AI985 LED 模块唤醒过程中不亮灯排查记录
- (4) 【FAQ1858】休眠唤醒重启问题一般排查思路
- (5) 【FAQ1500】LINUX 设备休眠唤醒回调卡死位置快速定位
- (6) 【FAQ1618】系统唤醒后模块工作异常的一般排查思路
- (7) 【FAQ795】进入休眠失败（休眠 abort）原因分析思路






著作权声明

版权所有 ©2024 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、、**全志科技**、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。